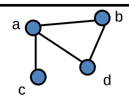
CSE 421
Algorithms

Graph Algorithms
Winter 2020
Lecture 7

Announcements

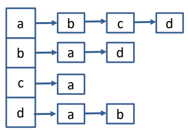
- Homework 3 Available
- My office hours today: 2:30-3:30 pm.

Graph Theory



	1	1	1
1		0	1
1	0		0
1	1	0	

Incidence Matrix



Adjacency List

- G = (V, E)
  - V: vertices, |V|= n
  - E: edges, |E| = m
- Undirected graphs
  - Edges sets of two vertices {u, v}
- Directed graphs
  - Edges ordered pairs (u, v)
- Path: $v_1, v_2, \ldots, v_k$, with (v_i, v_{i+1}) in E
  - Simple Path
  - Cycle
  - Simple Cycle
- Neighborhood
  - N(v)

Graph Algorithms (Review)

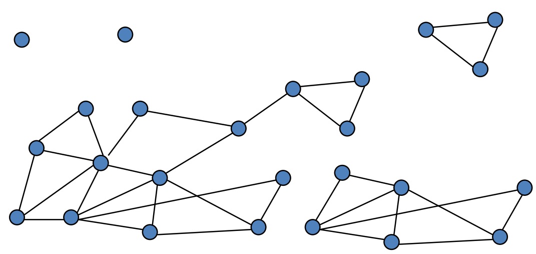
- Graph Search (Undirected or Directed graphs)
  - Find a path from s to t. O(n + m) time.
- Breadth First Search (Undirected) O(n+m) time
  - Non tree edges: Intra level edges or adjacent levels
- Depth First Search (Undirected) O(n+m) time
  - Non tree edges: Back edges
- Two coloring algorithm (Bipartite testing)
  - Constructed BFS and color levels alternating colors
  - Graph is bipartite iff no odd length cycles

Graph Connectivity

- An undirected graph is connected if there is a path between every pair of vertices x and y
- A connected component is a maximal connected subset of vertices

Connected Components

- Undirected Graphs

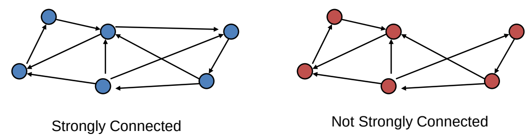


Computing Connected Components in O(n+m) time

- A search algorithm from a vertex v can find all vertices in v's component
- While there is an unvisited vertex v, search from v to find a new component

Directed Graphs

- A directed graph is strongly connected if for every pair of vertices x and y, there is a path from x to y, and there is a path from y to x

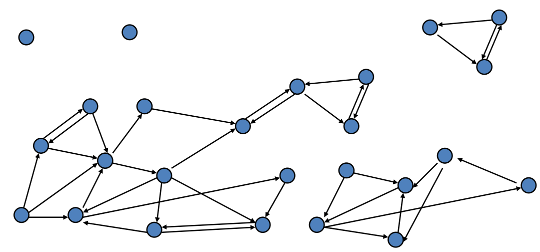


Testing if a graph is strongly connected

- Pick a vertex x
  - S_1 = { y | path from x to y }
  - S_2 = { y | path from y to x }
  - If $|S_1|$ = n and $|S_2|$ = n then strongly connected

Strongly Connected Components

A set of vertices C is a strongly connected component if C is a maximal strongly connected subgraph

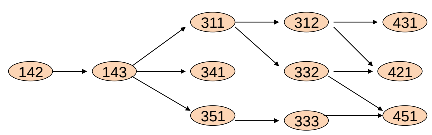


Strongly connected components can be found in O(n+m) time

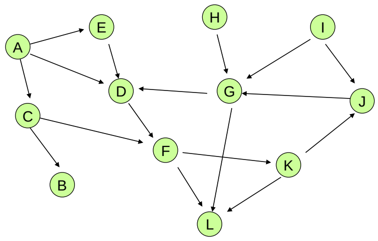
- But it's tricky!
- Simpler problem: given a vertex v, compute the vertices in v's scc in O(n+m) time

Topological Sort

- Given a set of tasks with precedence constraints, find a linear order of the tasks

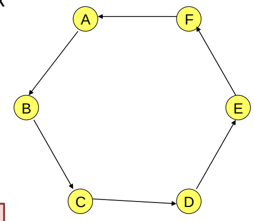


Find a topological order for the following graph



If a graph has a cycle, there is no topological sort

- Consider the first vertex on the cycle in the topological sort
- It must have an incoming edge



Definition: A graph is Acyclic if it has no cycles

Lemma: If a (finite) graph is acyclic, it has a vertex with in-degree 0

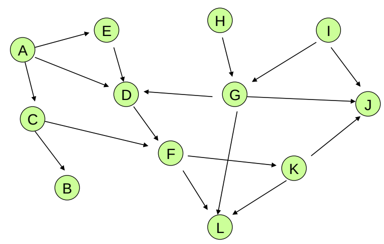
- Proof:
  - Pick a vertex v_1, if it has in-degree 0 then done
  - If not, let (v_2, v_1) be an edge, if v_2 has in-degree 0 then done
  - If not, let (v_3, v_2) be an edge . . .
  - If this process continues for more than n steps, we have a repeated vertex, so we have a cycle

Topological Sort Algorithm

While there exists a vertex v with in-degree 0

    Output vertex v

    Delete the vertex v and all out going edges

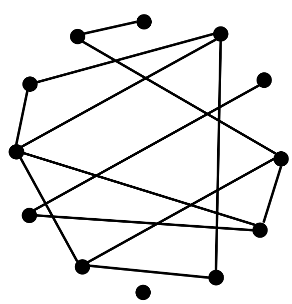


Details for O(n+m) implementation

- Maintain a list of vertices of in-degree 0
- Each vertex keeps track of its in-degree
- Update in-degrees and list when edges are removed
- m edge removals at O(1) cost each

Random Graphs

- What is a random graph?
- Choose edges at random
- Interesting model of certain phenomena
- Mathematical study
- Useful inputs for graph algorithms



Model of Random Graphs

- Undirected Graphs
  - Random Graph with n vertices and m edges, G_m
  - Random Graph with n vertices where each edge has probability p, G_p
  - Models are similar when $p = 2m / (n * (n – 1))$

```
for (int i = 0; i < n - 1; i++)
   for (int j = i + 1; j < n; j++)
       if (random.NextDouble() < p)
           AddEdge(i, j);
```